Herding cats: Modelling, simulation, testing, and data-mining for weak memory

Jade Alglave, Luc Maranget, Michael Tautschnig

Herding cats: Modelling, simulation, testing, and data-mining for weak memory

Jade Alglave
University College London
j.alglave@ucl.ac.uk

Luc Maranget
INRIA
luc.maranget@inria.fr

Michael Tautschnig
Queen Mary University of London
mt@eecs.qmul.ac.uk

Categories and Subject Descriptors B.3.2 [*Shared memory*]; C.0 [*Hardware/software interfaces*]

Keywords Concurrency, Weak Memory Models, Verification

There is a joke where a physicist and a mathematician are asked to herd cats. The physicist starts with an infinitely large pen which he reduces until it is of reasonable diameter yet contains all the cats. The mathematician builds a fence around himself and declares the outside to be the inside. Defining memory models is akin to herding cats: both the physicist's or mathematician's attitudes are tempting, but neither can go without the other.

When executing shared-memory concurrent programs, modern multiprocessors (e.g. Intel x86, IBM Power or ARM) exhibit behaviours (e.g. store buffering or load delaying) that contradict the programming model we have been taught at school, namely Lamport's Sequential Consistency (SC) [4]. Indeed, for performance reasons, multiprocessors implement *weak memory models*.

Ideally, we believe that these models would benefit from stating principles that underpin weak memory as a whole, not just one particular architecture or language. Not only would it be aesthetically pleasing, but it would allow more informed decisions on the design of high-level memory models, ease the conception and proofs of compilation schemes, and allow the reusability of simulation and verification techniques from one model to another.

We outline our work below – full details can be found in [1]. As foundation, we propose an axiomatic generic framework for modelling weak memory hardware. We have four axioms: (1) SC PER LOCATION expresses memory coherence; (2) NO THIN AIR defines the minimal causality constraints enforced by deployed hardware; (3) OBSERVATION reflects the ordering of writes induced by memory fences, as observed by an external thread; (4) PROPAGATION states how one can use memory fences to restore SC on top of a weak memory model.

We instantiate our framework for SC, TSO, C++ restricted to release-acquire atomics, Power, and ARM. For Power, we compare our model to a preceding operational model [5, 6] in which we found a flaw. To facilitate the comparison, we define in the Coq proof assistant an operational model that we show equivalent to our axiomatic model (see also www.cs.ucl.ac.uk/staff/j.alglave/cats).

PLDI '14, June 09–11, 2014, Edinburgh, United Kingdom.

http://dx.doi.org/10.1145/2594291.2594347

We also propose a model for ARM, based on extensive testing of ARM hardware, and its presumed proximity to Power hardware (see also diy.inria.fr/cats/model-arm). Our testing on this architecture revealed a behaviour later acknowledged as a bug by ARM, and more recently we discovered 31 additional anomalies, documented in our experimental reports at diy.inria.fr/cats/arm-anomalies.

To complement the dynamic analysis efforts, we implemented or improved a range of static analysis tools. First, we offer a new simulation tool (herd, see also diy.inria.fr/herd). Given a user-defined, concise specification, herd becomes a simulator for that model. The tool relies on an axiomatic description; this choice allows us to outperform all previous simulation tools. Second, for bounded model checking of software we confirm (echoing [2]) that verification time is vastly improved.

Finally, we put our models in perspective, in the light of empirical data obtained by analysing the C and C++ code of a Debian Linux distribution. We present our new static analysis tool, called mole (see also diy.inria.fr/mole), which explores a piece of code to find the weak memory idioms that it uses.

Acknowledgments

Our work [1] benefited from the help of numerous people. We thank Nikos Gorogiannis for suggesting that the extension of the input files for herd should be .cat and Daniel Kroening for infrastructure for running mole. We thank Mark Batty, Shaked Flur, Carsten Fuhs (even more so since we forgot to thank him in [2]), Matthew Hague, Tyler Sorensen, Gadi Tellez and Viktor Vafeiadis for their patient and careful comments on a draft. We thank our reviewers for their careful reading, comments and suggestions. We thank Arthur Guillon for his help with the simulator of [3], and Susmit Sarkar, Peter Sewell and Derek Williams for discussions on the Power model(s). Finally, this work would not have been the same without the discussions on related topics with Richard Bornat, Alexey Gotsman, Peter O'Hearn and Matthew Parkinson.

References

[1] J. Alglave, L. Maranget, and M. Tautschnig. Herding cats: Modelling, simulation, testing, and data-mining for weak memory. *TOPLAS*, 36 (2). To appear.

[2] J. Alglave, D. Kroening, and M. Tautschnig. Partial Orders for Efficient Bounded Model Checking of Concurrent Software. In *CAV*, 2013.

[3] G. Boudol, G. Petri, and B. Serpette. Relaxed Semantics of Concurrent Programming Languages. In *Express/SOS*, 2012.

[4] L. Lamport. How to Make a Correct Multiprocess Program Execute Correctly on a Multiprocessor. *IEEE Trans. Comput.*, 1979.

[5] S. Sarkar, P. Sewell, J. Alglave, L. Maranget, and D. Williams. Understanding Power multiprocessors. In *PLDI*, 2011.

[6] S. Sarkar, K. Memarian, S. Owens, M. Batty, P. Sewell, L. Maranget, J. Alglave, and D. Williams. Synchronising C/C++ and Power. In *PLDI*, 2012.